Universidad de los Andes

Concurrencia en Java

Generar varios threads con identificación

```
public class T extends Thread {
   private int id ;

   public T (int n) {
      id = n ;
   }
   …
   public void run () {
      // puede usar id
   }
   …
  for (int i=0; i < nThreads; i++) {
     new T(i).start () ;
  }
}
```

Universidad de los Andes

Concurrencia en Java

```
public class T extends Thread {
    private int id;

    public T (int n) {
       id = n ;
    }
    …
    public void run () {
      // puede usar id
    }
    …
    for (int i=0; i<nThreads; i++) {
        new T(i).start( ) ;
  }
}
```

52 Si nThreads = 10, ¿cuántos threads hay?

Universidad de los Andes

Concurrencia en Java

Buscar un valor en una matriz

```
public class T extends Thread {
    private static int valor ;
    private static int tamano ;
    private static int [][] M ;

    private int id ;

    public T (int i) {
      id = i ;
    }

    private static void inicializar( ) {
      // inicializa valor, tamano, M
    }
}
```

Universidad de los Andes

nearpod

Concurrencia en Java

Buscar un valor en una matriz

```
public class T extends Thread {
    private static int valor ;
    private static int tamano ;
    private static int [][] M ;

    private int id ;

    public T (int i) {
      id = i ;
    }

    private static void inicializar( ) {
      // inicializa valor, tamano, M
    }
}
```

Universidad de los Andes

Concurrencia en Java

Buscar un valor en una matriz

```
public void run (){
  int nElementos = M[id].length ;

  for (int j=0; j < nElementos; j++) {
    if (M[id][j] == valor) {
      System.out.println (id) ;
    }
  }
}
public static void main(String[] args) {
  T.inicializar () ;
  for (int i=0; i<tamano; i++)
    new T(i).start () ;
}
```

Universidad de los Andes

Concurrencia en Java

O en la misma clase (Depende de lo que necesitemos)

Buscar un valor en una matriz

Clase T

```
public void run () {
  int nElementos = M[id].length ;

  for (int j=0; j<nElementos; j++) {
    if (M[id][j] == valor) {
      System.out.println (id) ;
    }
  }
}
```

Clase Ppal

```
public static void main (String[] args) {
  T.inicializar () ;
  for (int i=0; i<n; i++)
    new T(i).start() ;
}
```

Universidad de los Andes

Concurrencia en Java

Interfaz runnable

```
public class R implements Runnable {
  …
  public void run () {
    // acciones del thread
  }
}
```

Creación y activación

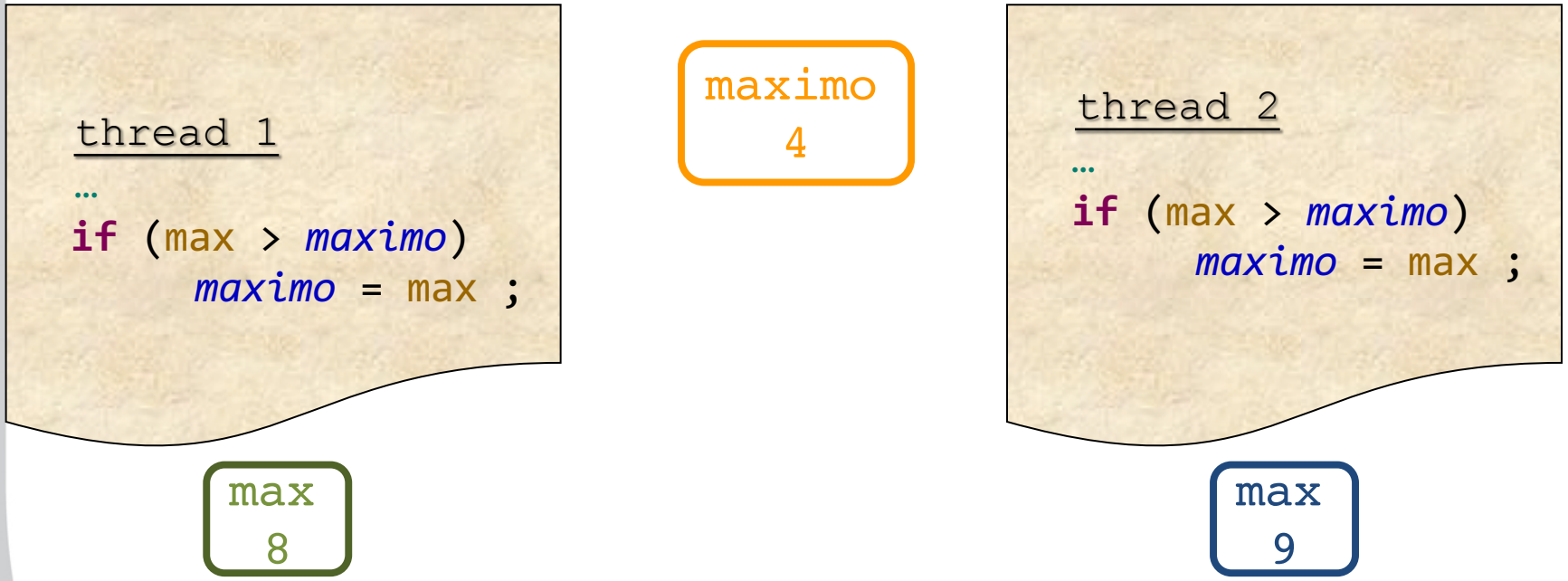
```
R r = new R () ;  //Creación del objeto
Thread t = new Thread (r);  //Creación del thread
…
t.start( ) ;  //Activación
```

2 threads !!

Universidad de los Andes

Sincronización

- Acceso concurrente a los datos



Universidad de los Andes

Sincronización

- Acceso concurrente a los datos

```
thread 1
…
load R1, maximo
load R2, max
cmp R2, R1
jle continuar
  store maximo, R2
continuar:
```

maximo
4

```
thread 2
…
load R1, maximo
load R2, max
cmp R2, R1
jle continuar
  store maximo, R2
continuar:
```

max
8

max
9

Universidad de los Andes

Sincronización

- Orden en el acceso a los datos

```
thread
load R1, suma
…
add R1, R2
store suma, R1
```

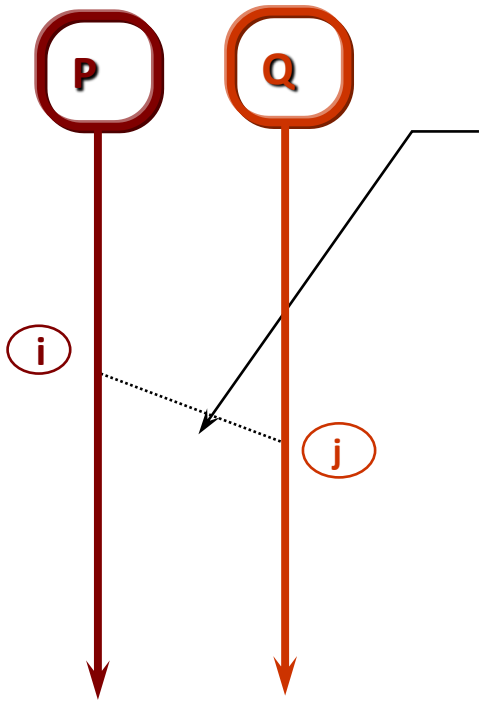
```
main
load R1, suma
load R2, total
add R2, R1
store total, R2
```

Sincronización

- Monitores
- Eventos, variables de condición
- Semáforos
  - Binarios
- Barreras

Universidad de los Andes

Sincronización



Ordenamiento

Restricción sobre el momento de ejecución de dos eventos:

$T_i \neq T_j$ (exclusión mutua)

$T_i = T_j$ (encuentro)

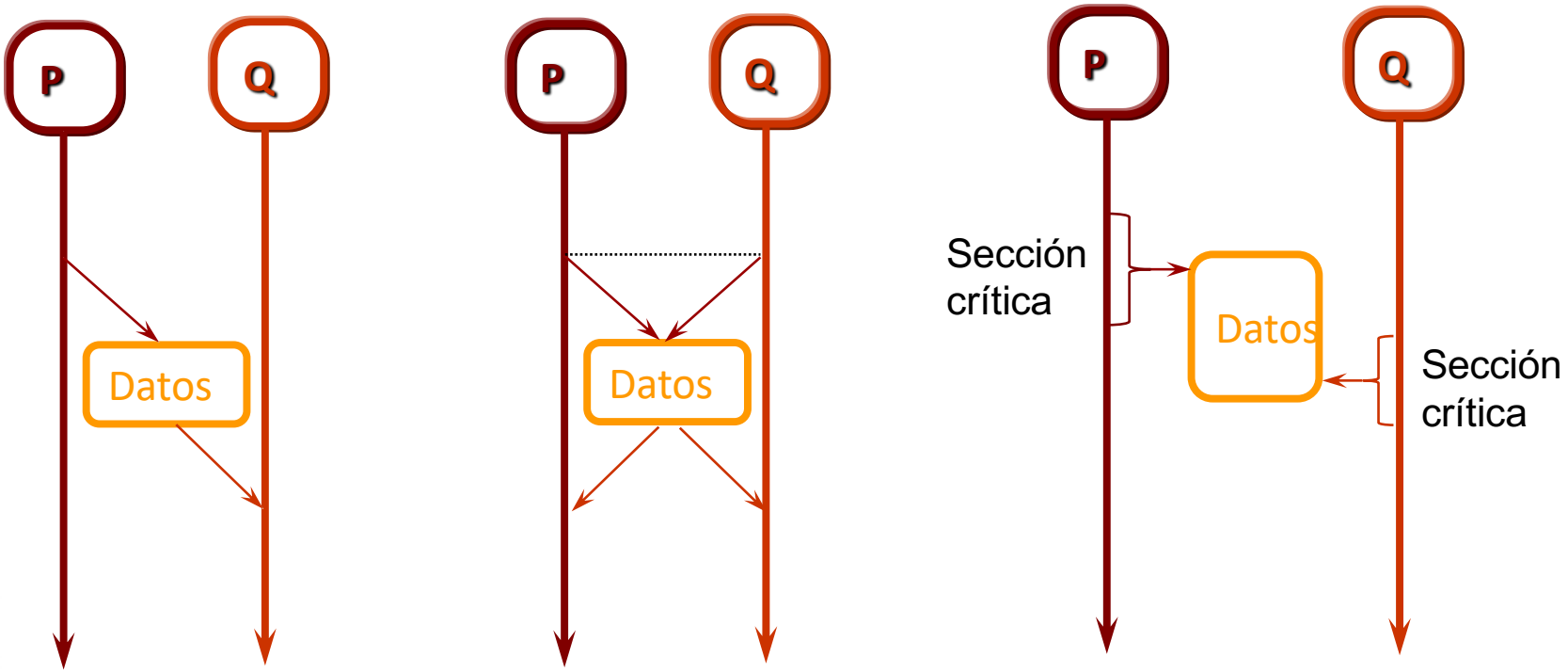
$T_i < T_j$ (señalamiento)

T_i ? T_j (no determinismo)

Universidad de los Andes

- Encuentro
- Exclusión mutua
- Señalamiento

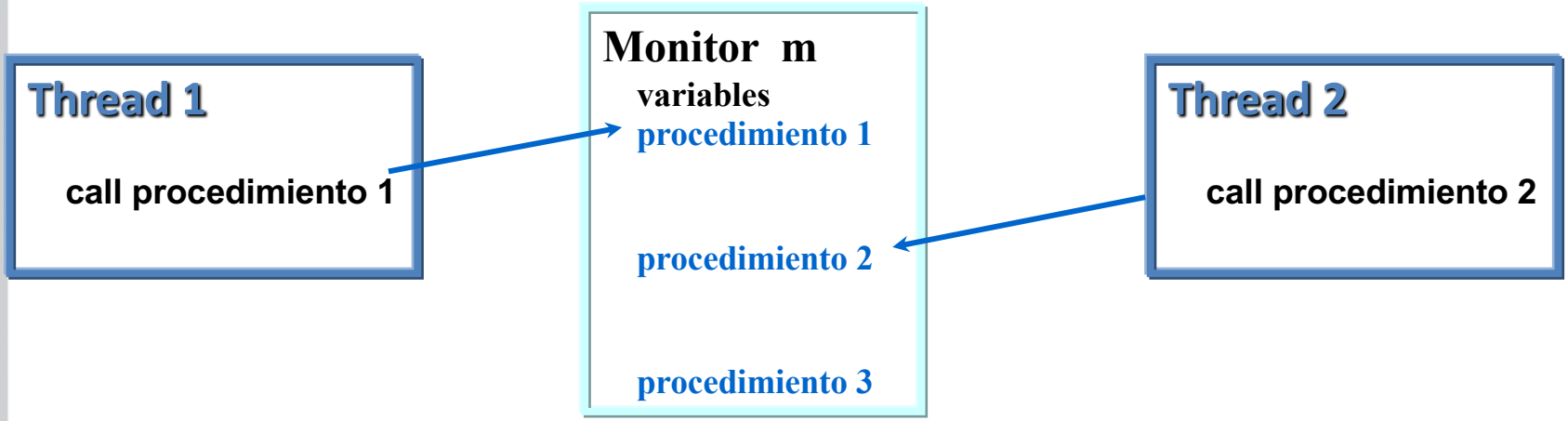
Sincronización



Universidad de los Andes

Sincronización - exclusión mutua

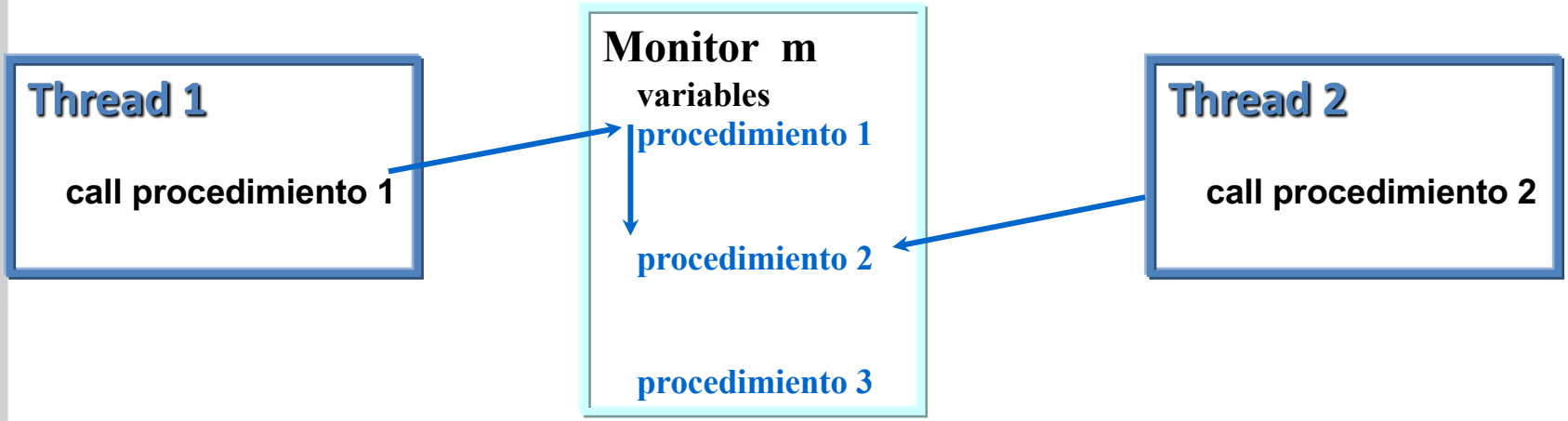
- Monitores:



Universidad de los Andes

Sincronización - exclusión mutua

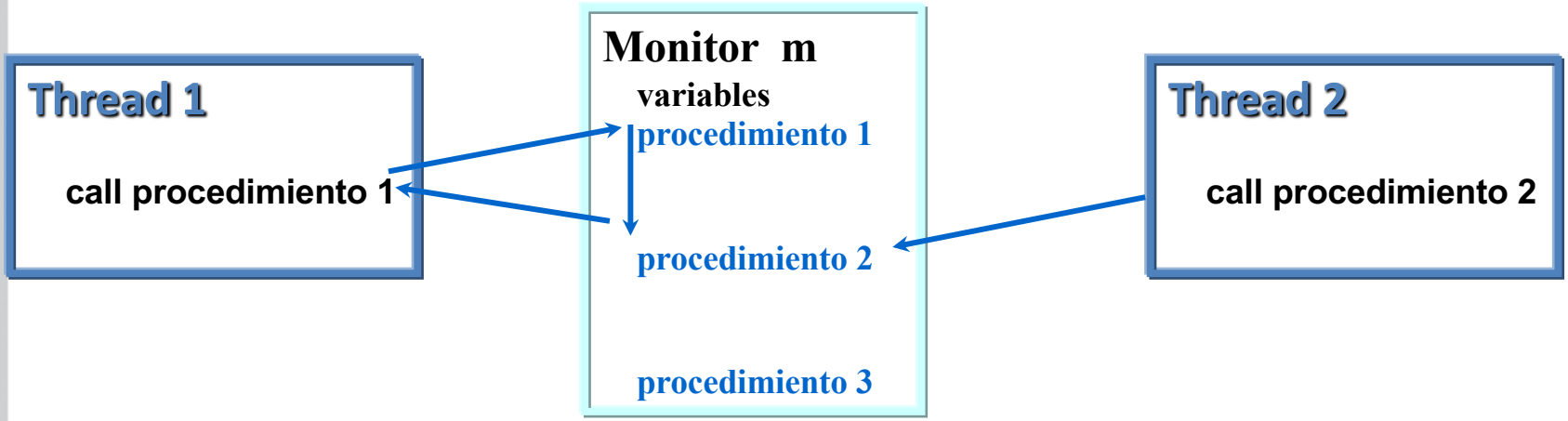
- Monitores:



Universidad de los Andes

Sincronización - exclusión mutua

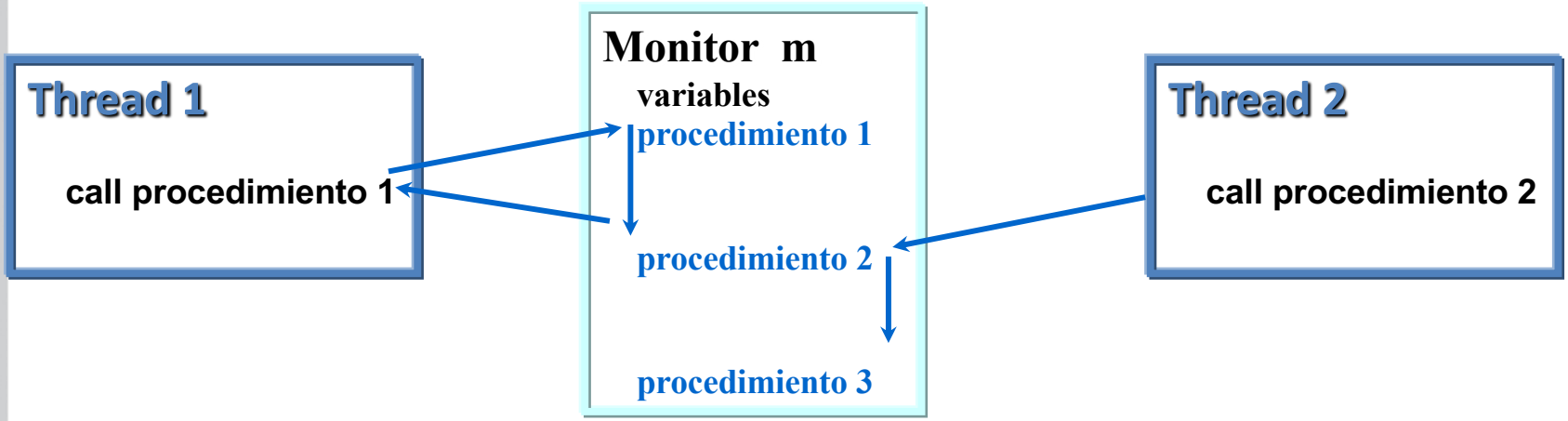
- Monitores:



Universidad de los Andes

Sincronización - exclusión mutua

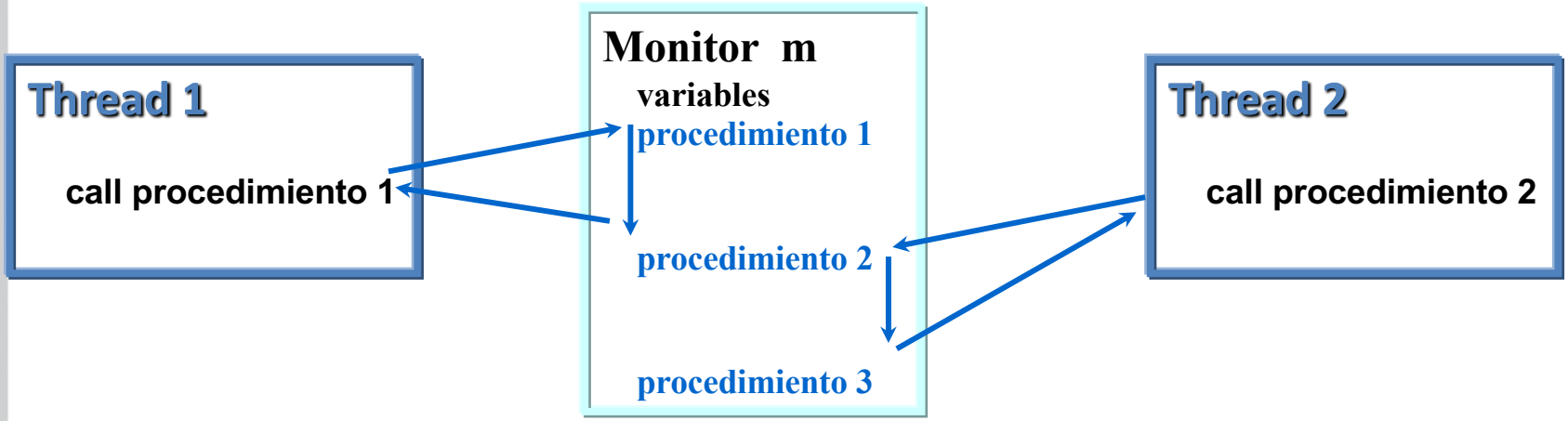
- Monitores:



Universidad de los Andes

Sincronización - exclusión mutua

- Monitores:



Universidad de los Andes

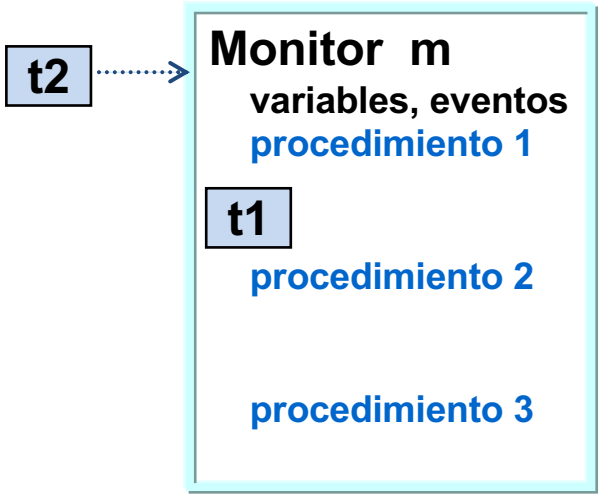
Sincronización - exclusión mutua

- Monitores:

Universidad de los Andes

Sincronización - exclusión mutua

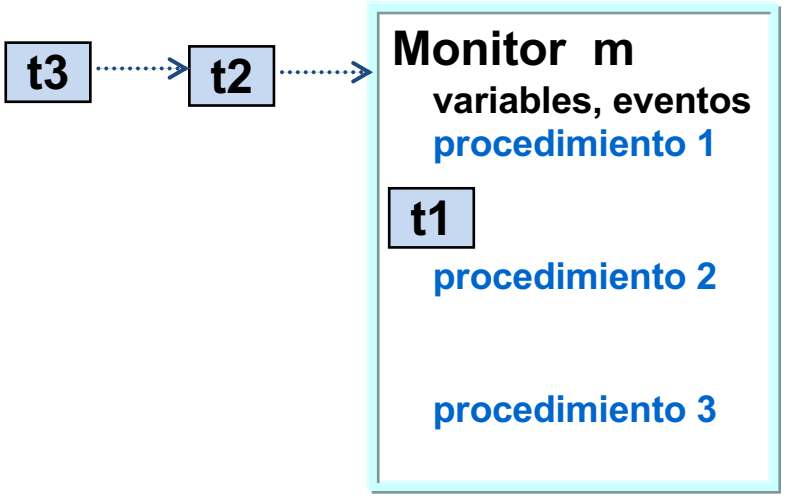
- Monitores:



Universidad de los Andes

Sincronización - exclusión mutua

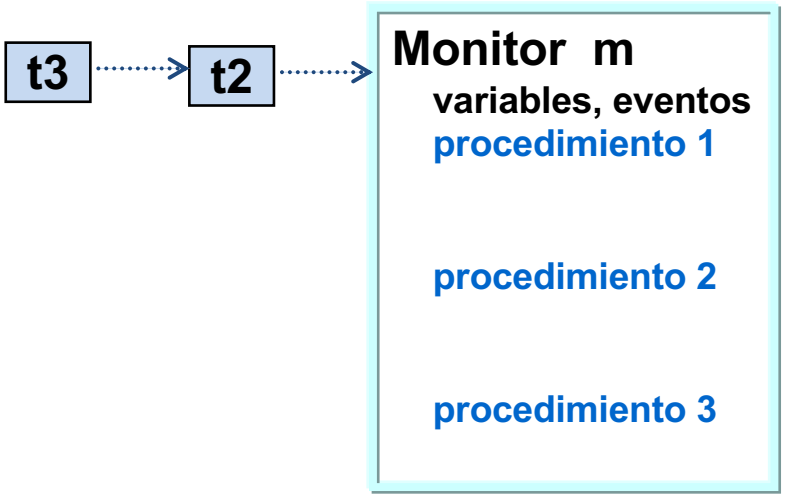
- Monitores:



Universidad de los Andes

Sincronización - exclusión mutua

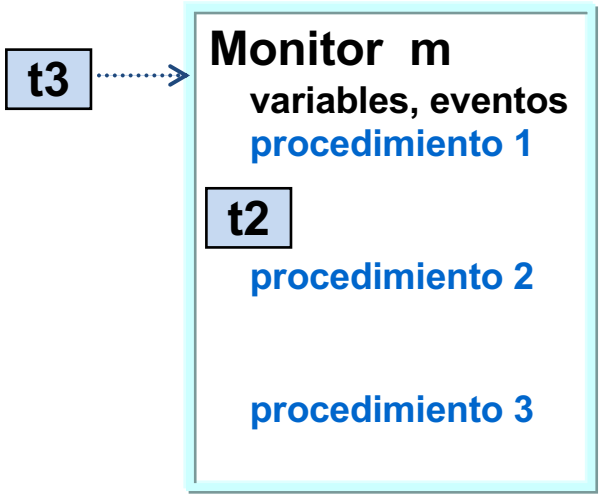
- Monitores:



Universidad de los Andes

Sincronización - exclusión mutua

- Monitores:



Universidad de los Andes

Sincronización - exclusión mutua

- Java: monitores

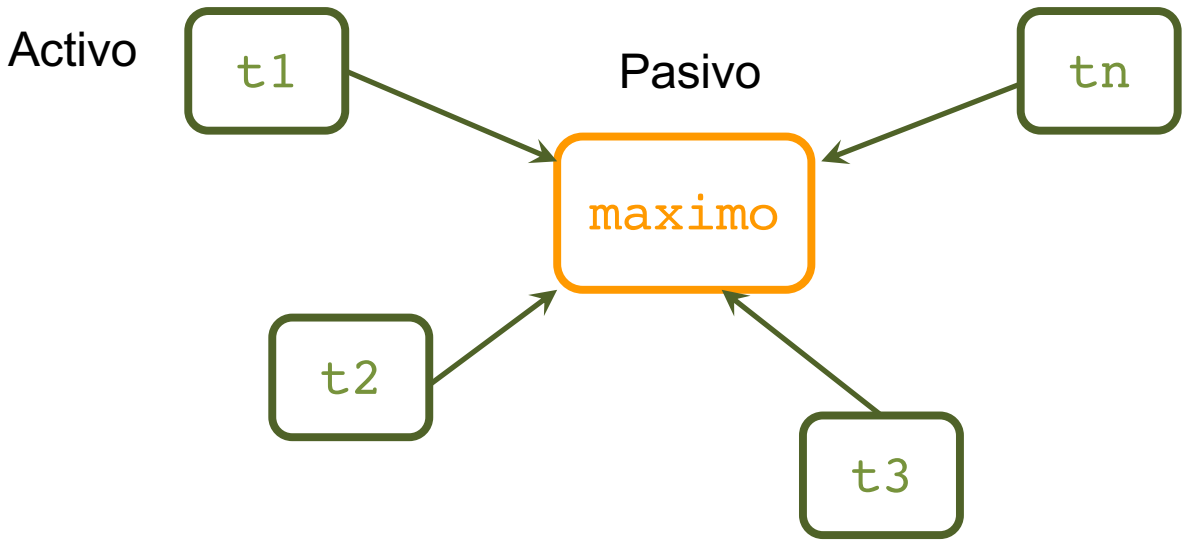
```
public class C {
  // atributos
  public synchronized void m1 (…) {
    …
  }
  public synchronized void m2 (…) {
    …
  }
  …
  public void mn (…) {
    …
  }
```

Sincronización de objetos !!

Universidad de los Andes

Sincronización - exclusión mutua

- Java – ejemplo



Universidad de los Andes

Sincronización - exclusión mutua

- Java – ejemplo

```
public class Maximo {

    private int maximo = 0 ;

    public synchronized void anotar (int n) {
        if (n > maximo)
            maximo = n ;
    }

    public synchronized int darMaximo () {
        return maximo ;
    }
}
```

La clase que extienda de Thread debe tener referencia a ese objeto

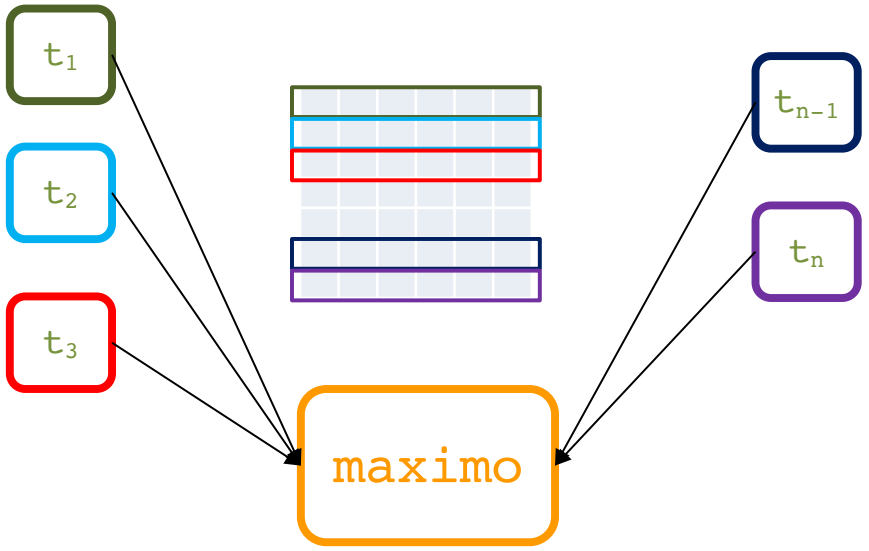
Universidad de los Andes

```
public class T extends Thread {

    private  static Maximo oMax = new Maximo () ;
    private int num = 0 ;

    public T (int n) {
        num = n ;
    }

    public void run () {
        oMax.anotar (num) ;
    }

    public static void main(String[] args) {

        for ( int i = 0; i < 10; i++)
            new T(i).start() ;

        System.out.println ("El máximo es: " + oMax.darMaximo()) ;
    }
}
```

Universidad de los Andes

Sincronización - exclusión mutua

- Java – ejercicio . Encontrar el máximo de la matriz:
  - Thread por fila
  - El último en ejecutar imprime el resultado
  - `anotar` actualiza e informa si es el último



Cuántas clases?
Quién conoce `nThreads`?
Dónde está `anotar`?

```
System.out.println ("El máximo es " … );
```

Usaremos dos clases:
- Clase T
- Clase Max

Universidad de los Andes

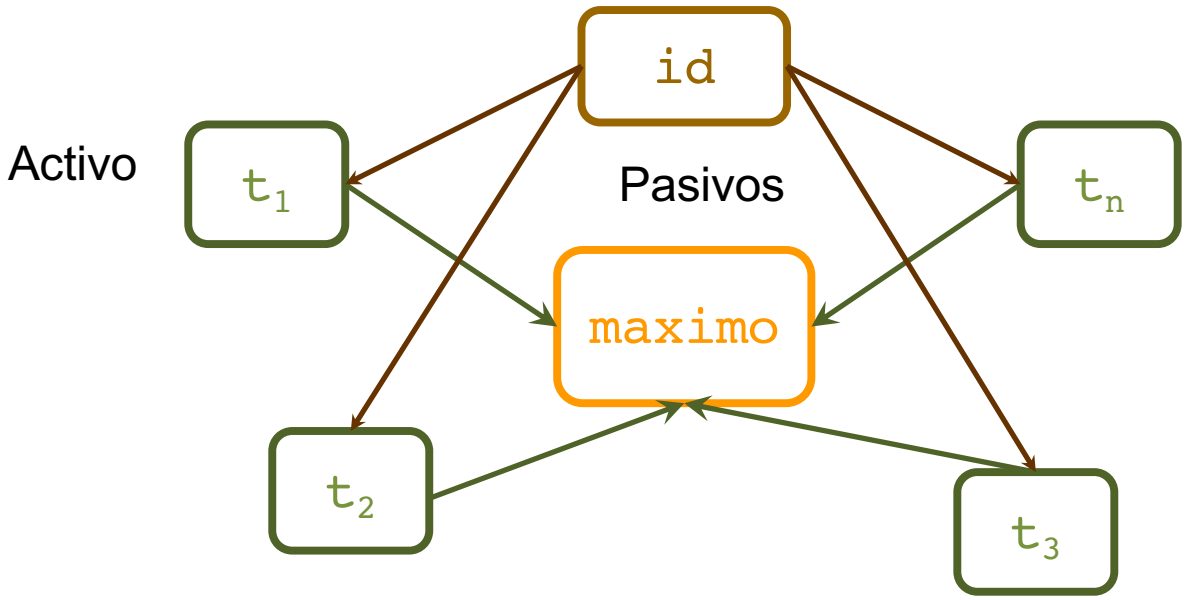
Sincronización - exclusión mutua

- Java – ejemplo . Repetir ejercicio anterior, pero ahora el `main` no pasa el identificador entero como parámetro. En lugar de esto:
  - Se tiene un objeto `id` que reparte identificadores. Este objeto tiene un método `darId()` que retorna los números de 0 a n-1 en secuencia.
  - Cuando el `main()` crea los threads, les pasa como parámetro una referencia a `id`.
  - Cada thread invoca `darId()` para tener su identificador (el número de fila que le toca a él). Después, proceden igual que en el caso anterior.

Universidad de los Andes

Sincronización - exclusión mutua

- Java – ejemplo



Universidad de los Andes

Código calcular máximo con asignador de fila

```
public class Maximo { // Igual que el anterior
   private int cont, maximo, nThreads ;

   public Maximo (int numT) {
      nThreads = numT ;
      maximo = 0 ;
      cont = 0 ;
   }
   public synchronized boolean anotar (int n){
      if (n > maximo) maximo = n ;
         return
           (++cont == nThreads) ? true: false;
    }
}

public class Identificar {
   private int numId ;

   public Identificar () {
      numId = 0 ;
   }
   public synchronized darNumId () {
      return numId ++ ;
   }
}
```

```
public class T extends Thread {
   private static Identificar objId ;
   private static Maximo m ;
   private static int [][] mat ;

   public void run () {
      int locMax = 0 ;
      int id = objId.darNumId () ;
      for (int j=0; j<mat[id].lenght; j++)
          if (mat[id][j] > locMax)
              locMax = mat[id][j] ;
      if (m.anotar(locMax))
         System.out.println
            ("Max: " + m.darMaximo ()) ;
   }
   public static void main (String[] args) {
      int n = 10 ;
      inicializar () ;  // crea/llena mat
      objId = new Identificar () ;
      m = new Maximo (n) ;
      for (int i=0; i<n; i++)
         new T().start() ;
   }
}
```

Universidad de los Andes

Sincronización - exclusión mutua

- Java – ejercicio . Resolver de nuevo el ejercicio del máximo, pero con las siguientes características:
  - Se quiere encontrar el máximo, pero de un vector de M posiciones .
  - Se quiere generar una cierta cantidad de threads tal, que a cada uno le toque procesar N elementos del vector.
  - M no necesariamente es divisible por N, luego a uno de los thread le pueden tocar menos elementos.